Dialogue Systems

NPFL123 Dialogové systémy

9. Neural Policies & Natural Language Generation

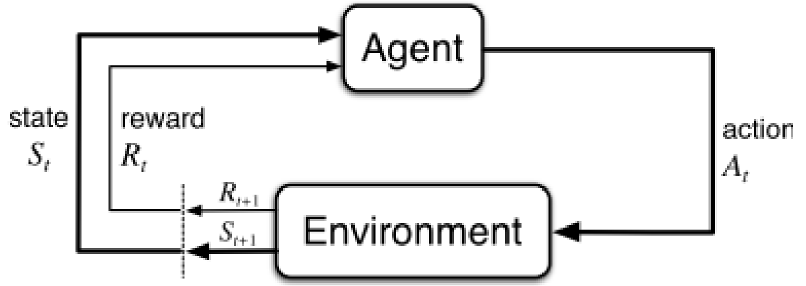
Ondřej Dušek & Vojtěch Hudeček & Jan Cuřín

http://ufal.cz/npfl123

14. 4. 2020

Deep Reinforcement Learning

- Exactly the same as "plain" RL
  - agent & environment, actions & rewards
  - Markov Decision Process
- **"deep" = part of the agent is handled by a NN**
  - value function (typically Q)
  - policy
- NN = function approximation approach
  - such as REINFORCE / policy gradients
  - NN → complex non-linear functions
- assuming huge state space
  - much fewer weights than possible states
  - update based on one state changes many states



(Sutton & Barto, 2018)

Value Function Approximation

- Searching for approximate $V(s)$ or $Q(s,a)$
  - exact values are too big to enumerate in a table
  - **parametric approximation $V(s;\boldsymbol{\theta})$ or $Q(s,a;\boldsymbol{\theta})$**
- Regression: **Mean squared value error**
  - weighted over states' importance
  - useful for gradient descent
  - → ~ **any supervised learning approach possible**
    - not all work well though
- MC = stochastic gradient descent
- TD is **semi-gradient** (not true gradient descent)
  - ← using current weights in target estimate
  - we still want TD over MC for speed
  - guaranteed convergence for linear approximations
  - unstable for NNs!

our estimate

states' importance weight
(probability distribution)

$$\overline{\mathrm{VE}}(\boldsymbol{\theta}) \coloneqq \sum_{s\in\mathcal{S}} \mu(s)\big(V_\pi(s) - V(s,\boldsymbol{\theta})\big)^2$$

target value
(which we don't have!)
→ using R_t in MC
→ using $r_{t+1} + \gamma V(s',\boldsymbol{\theta})$

Deep Q-Networks (Mnih et al., 2013, 2015)

- Q-learning with function approximation
  - Q function represented by a neural net
- Causes of poor convergence in basic Q-learning with NNs:
  a) SGD is unstable
  b) correlated samples (data is sequential)
  c) TD updates aim at a moving target (using Q in computing updates to Q)
  d) scale of rewards & Q values unknown → numeric instability
- Fixes in DQN:
  a) minibatches (updates by averaged n samples, not just one)
  b) experience replay
  c) freezing target Q function
  d) clipping rewards

cool!

common NN tricks

DQN tricks ~ making it more like supervised learning

- **Experience replay** – break correlated samples
  - run through some episodes (dialogues, games…)
  - store all tuples (s, a, r', s') in a buffer
  - for training, don't update based on most recent moves – use buffer
    - sample minibatches randomly from the buffer
  - overwrite buffer as you go, clear buffer once in a while
  - only possible for off-policy

"generate your own 'supervised' training data"

$$\text{loss} \coloneqq \mathbb{E}_{(s,a,r',s')\in\text{buf}}\left[\left(r' + \gamma \max_{a'} Q\left(s', a'; \overline{\boldsymbol{\theta}}\right) - Q(s, a; \boldsymbol{\theta})\right)^2\right]$$

- **Target Q function freezing**
  - fix the version of Q function used in update targets
    - have a copy of your Q network that doesn't get updated every time
  - once in a while, copy your current estimate over

"have a fixed target, like in supervised learning"

DQN algorithm

- initialize $\boldsymbol{\theta}$ randomly
- initialize replay memory D (e.g. play for a while using current $Q(\boldsymbol{\theta})$)
- repeat over all episodes:
  - for episode, set initial state s
    - select action a from ϵ-greedy policy based on $Q(\boldsymbol{\theta})$
    - take a, observe reward r' and new state s'
    - store (s, a, r', s') in D
    - $s \leftarrow s'$

  storing experience

  often
  - once every k steps:
    - sample a batch B of random (s, a, r', s')'s from D
    - update $\boldsymbol{\theta}$ using loss $\mathbb{E}_{(s,a,r',s') \in B} \left[\left(r' + \gamma \max_{a'} Q\left(s', a'; \overline{\boldsymbol{\theta}}\right) - Q(s, a; \boldsymbol{\theta}) \right)^2 \right]$

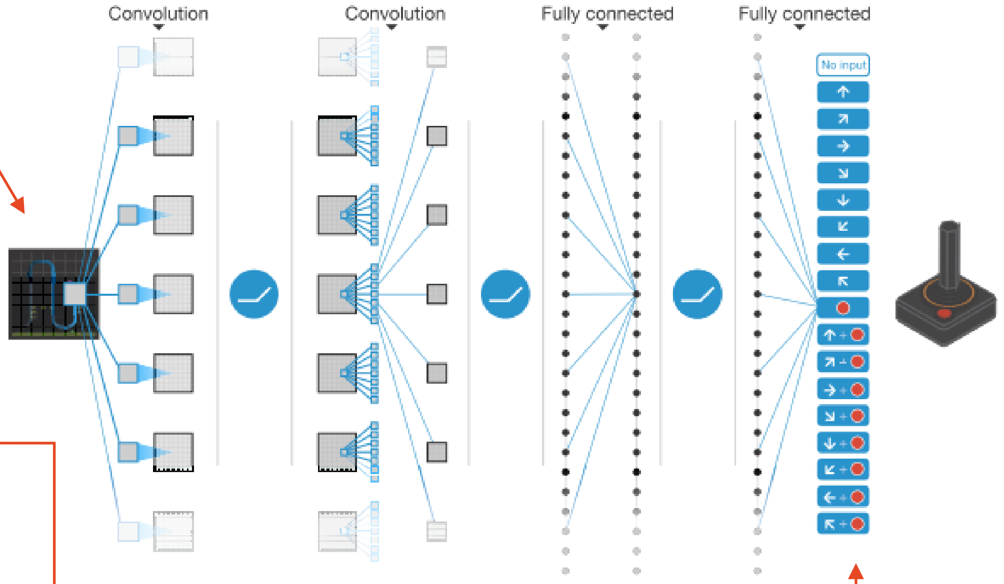
  "replay"
  a. k. a. training

  rarely
  - once every λ steps:
    - $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$

DQN for Atari

input: Atari 2600 screen,
downsized to 84x84 (grayscale)
4 last frames

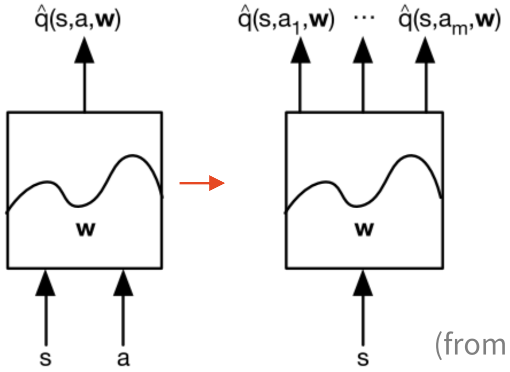
(Mnih et al., 2015)



values for all actions
(joystick moves)

- 4-layers:
  - 2x CNN
  - 2x fully connected with ReLU activations
- Another trick:
  - output values for all actions at once
    - ~ vector $\boldsymbol{Q}(s)$ instead of $Q(s, a)$
    - a is not fed as a parameter
  - faster computation
- Learns many games at human level
  - with the same network structure
  - no game-specific features

https://youtu.be/V1eYniJ0Rnk?t=18



(from David Silver's slides)

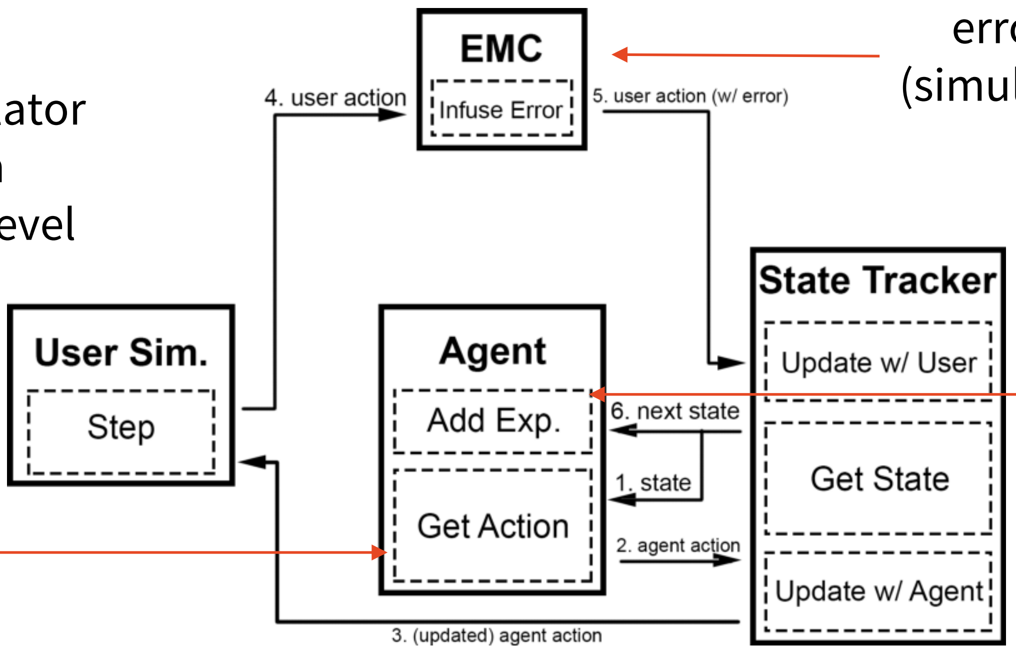
DQN for Dialogue Systems

(Li et al., 2017)
https://arxiv.org/abs/1703.01008
https://github.com/MiuLab/TC-Bot

- a simple DQN can drive a dialogue system's action selection
  - DQN is function approximation – works fine for POMDPs
  - no summary space tricks needed here

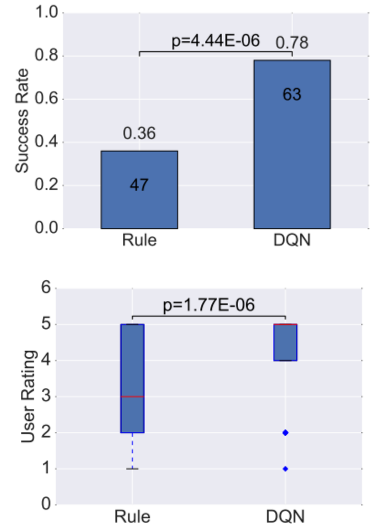
rule-based simulator
with agenda
running on DA level

DQN – feed-forward,
1 hidden ReLU layer



error model controller
(simulating ASR/NLU noise)

replay memory
initialized using a
simple handcrafted policy

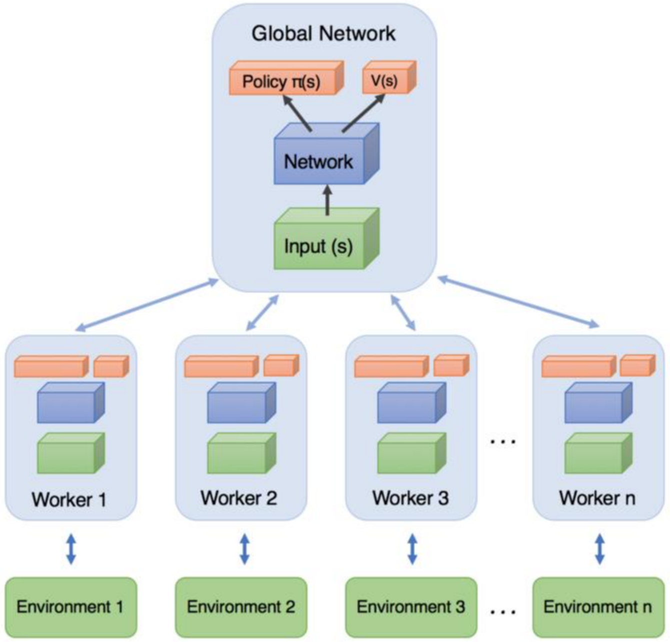


movie ticket booking:
better than rule-based

Policy Networks

policy gradient theorem
guarantees convergence

- Learning policy directly – **policy network**
  - can work better than Q-learning
  - NN: input = state, output = prob. dist. over actions
  - actor-critic: network predicts both π and V/Q
- Training can't use/doesn't need the DQN tricks
  - just REINFORCE with baseline / actor-critic
    - reward – baseline = **advantage**
  - these are on-policy → no experience replay
    - minibatches used anyway
  - extension: parallel training (A3C algorithm)
    - sample in multiple threads, gather gradients
    - better speed, more diverse experience



https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

Natural Language Generation

- conversion of **system action semantics → text** (in our case)
- NLG output is well-defined, but input is not:
  - DAs
  - any other semantic formalism
  - database tables
  - raw data streams
  - user model
  - dialogue history

can be any kind of knowledge representation

e.g. “user wants short answers”

e.g. for referring expressions, avoiding repetition

- general NLG objective:
  - **given input & communication goal**
  - **create accurate + natural, well-formed, human-like text**
- additional NLG desired properties:
  - variation
  - simplicity
  - adaptability

NLG Use Cases

- **dialogue systems**
  - very different for task/non-task-oriented/QA systems
- **standalone**
  - data-to-text
  - short text generation for web & apps
    - weather, sports reports
    - personalized letters
- **machine translation**
  - now mostly integrated end-to-end
  - formerly not the case
- **summarization**

NLG Subtasks (textbook pipeline)

Inputs

- ↓ **Content/text/document planning**
  - content selection according to communication goal
  - basic structuring & ordering

Content plan

- ↓ **Sentence planning/microplanning**
  - aggregation (facts → sentences)
  - lexical choice
  - referring expressions

Sentence plan

- ↓ **Surface realization**
  - linearization according to grammar
  - word order, morphology

Text

deciding what to say

deciding how to say it

typically handled by dialogue manager in dialogue systems

organizing content into sentences & merging simple sentences

e.g. *restaurant* vs. *it*

this is needed for NLG in dialogue systems

NLG Implementations

- **Few systems implement the whole pipeline**
  - All stages: mostly domain-specific data-to-text, standalone
    - e.g. weather reports
  - Dialogue systems: just sentence planning + realization
  - Systems focused on content + sentence planning with trivial realization
    - frequent in DS: focus on sentence planning, trivial or off-the-shelf realizer
  - Surface realization only
    - requires very detailed input
    - some systems: just ordering words
- **Pipeline vs. end-to-end approaches**
  - planning + realization in one go – popular for neural approaches
  - pipeline: simpler components, might be reusable (especially realizers)
  - end-to-end: no error accumulation, no intermediate data structures

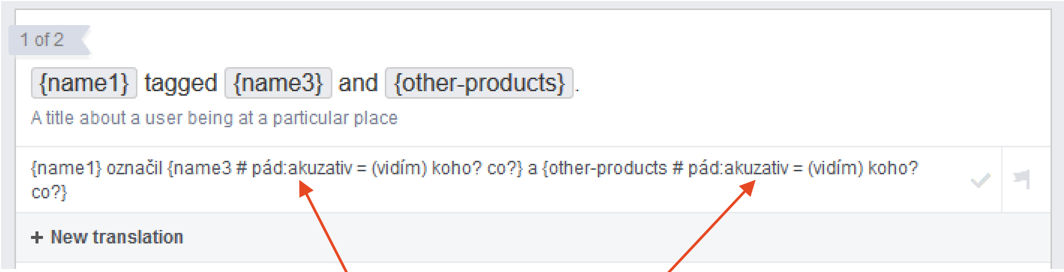
NLG Basic Approaches

- **canned text**
  - most trivial – completely hand-written prompts, no variation
  - doesn’t scale (good for DTMF phone systems)
- **templates**
  - “fill in blanks” approach
  - simple, but much more expressive – covers most common domains nicely
  - can scale if done right, still laborious
  - most production dialogue systems
- **grammars & rules**
  - grammars: mostly older research systems, realization
  - rules: mostly content & sentence planning
- **machine learning**
  - modern research systems
  - pre-neural attempts often combined with rules/grammar
  - RNNs made it work *much* better

Template-based NLG

- Most common in dialogue systems
  - especially commercial systems
- Simple, straightforward, reliable
  - custom-tailored for the domain
  - complete control of the generated content
- Lacks generality and variation
  - difficult to maintain, expensive to scale up
- Can be enhanced with rules
  - e.g. articles, inflection of the filled-in phrases
  - template coverage/selection rules, e.g.:
    - select most concrete template
    - cover input with as few templates as possible
    - random variation

(Facebook, 2015)



(Facebook, 2019)

inflection rules

```
'iconfirm(to_stop={to_stop})&iconfirm(from_stop={from_stop})':
    "Alright, from {from_stop} to {to_stop},",

'iconfirm(to_stop={to_stop})&iconfirm(arrival_time_rel="{arrival_time_rel}")':
    "Alright, to {to_stop} in {arrival_time_rel},",

'iconfirm(arrival_time="{arrival_time}")':
    "You want to be there at {arrival_time},",

'iconfirm(arrival_time_rel="{arrival_time_rel}")':
    "You want to get there in {arrival_time_rel},",
```

(Alex public transport information rules)
https://github.com/UFAL-DSG/alex

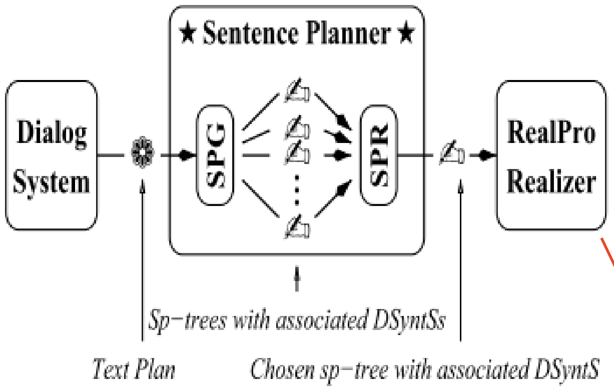
Trainable Sentence Planning:
Overgenerate & Rerank

(Walker et al., 2001)
https://www.aclweb.org/anthology/N01-1003

- Assuming you have a flexible handcrafted planner
  - underspecified grammar
  - rules with multiple options…
- Generate multiple outputs
- Select the best one
  - train just the selection – learning to rank
  - any supervised approach possible
    - a) “top” = 1, “not top” = 0
    - b) loss incurred by relative scores
      loss = max(0, “not top” – “top”)

this takes time!

SpoT trainable planner
(RankBoost ranking)



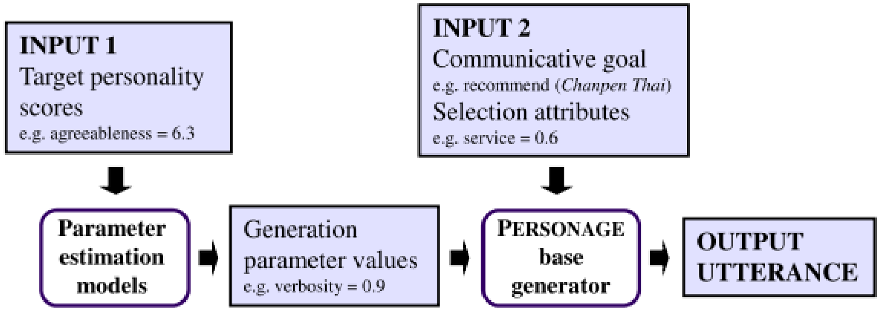
input DA

implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time)

Human RankBoost

Alt	Realization	H	RB
0	What time would you like to travel on September the 1st to Dallas from Newark?	5	.85
5	Leaving on September the 1st. What time would you like to travel from Newark to Dallas?	4.5	.82
8	Leaving in September. Leaving on the 1st. What time would you, traveling from Newark to Dallas, like to leave?	2	.39

Trainable Sentence Planning: Parameter Optimization



PERSONAGE-PE:
generation with
Big Five personality traits

- Assuming you have a flexible handcrafted planner
  - + one that has **configurable parameters**, for e.g.:
    - sentence aggregation
    - fillers
    - lexical choices
- Train the best parameters for your task
  - generate under different settings
  - annotate the outputs with linguistic features
  - learn classifiers: linguistic features → generator settings
    - any supervised learning
    - can predict the settings jointly/independently

I see, oh Chimichurri Grill is a latin american place with sort of poor atmosphere. Although it doesn't have rather nasty food, its price is 41 dollars. I suspect it's kind of alright.	extra=2.50 ems=4.50 agree=3.50 consc=4.75 open=4.25
Did you say Ce-Cent'anni? I see, I mean, I would consider it because it has friendly staff and tasty food, you know buddy.	extra=4.75 ems=5.00 agree=6.25 consc=6.25 open=5.25

extraversion
emotional stability
agreeableness
conscientiousness
openness to experience

(Mairesse & Walker, 2008; 2011)
https://www.aclweb.org/anthology/P08-1020
https://www.aclweb.org/anthology/J11-3002

Grammar-based realizers

- Various grammar formalisms
  - production / unification rules in the grammar
- typically general-domain, reusable
- **KPML** – multilingual
  - systemic functional grammar
- **FUF/SURGE** – English
  - functional unification grammar

KPML sentence plan
for *A dog is in the park.*

```
(l0 / spatial-locating
    :speechact (a0 / assertion :polarity positive
                               :speaking-time t0)
    :reference-time-id t0
    :event-time (t0 / time)
    :theme d0
    :domain (d0 / object :lex dog
                 :identifiability-q notidentifiable)
    :range (p0 / three-d-location :lex park
                 :identifiability-q identifiable))
```

(Bateman, 1997)
http://www.academia.edu/download/3459017/bateman97-jnle.pdf

FUF/SURGE input and output

Input Specification (I_1):

$$\begin{bmatrix} cat & clause \\ process & \begin{bmatrix} type & composite \\ relation & possessive \\ lex & \text{"}hand\text{"} \end{bmatrix} \\ partic & \begin{bmatrix} agent & \begin{bmatrix} cat & pers\_pro \\ gender & feminine \end{bmatrix} \\ affected & \boxed{1} \begin{bmatrix} cat & np \\ lex & \text{"}editor\text{"} \end{bmatrix} \\ possessor & \boxed{1} \\ possessed & \begin{bmatrix} cat & np \\ lex & \text{"}draft\text{"} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Output Sentence (S_1): "She hands the draft to the editor"

(Elhadad & Robin, 1996)
https://academiccommons.columbia.edu/doi/10.7916/D83T9RG1/download

Grammar-based Realizers: OpenCCG

- **OpenCCG** – English
  - combinatory categorial grammar
  - reuse/reverse of CCG parser
    - (reverse) lexical lookup
    - combination according to grammar – dynamic programming
  - statistical enhancements

OpenCCG input for flight information

```
be [tense=pres info=rh id=n1]
<Arg> flight [num=sg det=the info=th id=f2]
      <HasProp> cheapest [kon=+ id=n2]
<Prop> has-rel [id=n3]
       <Of> f2
       <Airline> Ryanair [kon=+ id=n4]
```

(Moore et al., 2004)
http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-155.pdf

$@_x(\mathbf{man} \wedge \langle \text{GenRel}\rangle(e \wedge \mathbf{see} \wedge \langle \text{Tense}\rangle \mathbf{past}$
$\wedge \langle \text{Act}\rangle(b \wedge \mathbf{Bob}) \wedge \langle \text{Pat}\rangle x))$

$0 : @_x\mathbf{man},\ 1 : @_x\langle \text{GenRel}\rangle e,\ 2 : @_e\mathbf{see}$
$3 : @_e\langle \text{Tense}\rangle\mathbf{past},\ 4 : @_e\langle \text{Act}\rangle b$
$5 : @_e\langle \text{Pat}\rangle x,\ 6 : @_b\mathbf{Bob}$

OpenCCG input

$\{2,3,4,5\}\ \{e,b,x\}$
saw $\vdash (s_{e,fin}\backslash np_b)/np_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{Tense}\rangle\mathbf{past} \wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x$

$\{2,4,5\}\ \{e,b,x\}$
see $\vdash (s_{e,nonfin}\backslash np_b)/np_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x$

$\{1\}\ \{e,x\}$
that $\vdash (n_x\backslash n_x)/(s_{e,fin}\backslash np_x) : @_x\langle \text{GenRel}\rangle e$

$\{1\}\ \{e,x\}$
that $\vdash (n_x\backslash n_x)/(s_{e,fin}/np_x) : @_x\langle \text{GenRel}\rangle e$

OpenCCG lexical lookup

(White & Baldridge, 2003)
https://www.aclweb.org/anthology/W03-2316

Bob $\vdash s_t/(s_t\backslash np_b) : @_b\mathbf{Bob}$

to see $\vdash (s_{e,inf}\backslash np_b)/np_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x$

Bob saw $\vdash s_{e,fin}/np_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{Tense}\rangle\mathbf{past}$
$\wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x \wedge @_b\mathbf{Bob}$

Bob to see $\vdash s_{e,inf}/np_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x \wedge @_b\mathbf{Bob}$

man that Bob saw $\vdash n_x$:
$@_x\mathbf{man} \wedge @_x\langle \text{GenRel}\rangle e$
$\wedge @_e\mathbf{see} \wedge @_e\langle \text{Tense}\rangle\mathbf{past}$
$\wedge @_e\langle \text{Act}\rangle b \wedge @_e\langle \text{Pat}\rangle x \wedge @_b\mathbf{Bob}$

OpenCCG parsing (combinatory rules)

Procedural realizer: SimpleNLG

- A simple Java API
  - “do-it-yourself” style – only cares about the grammar
  - input needs to be specified precisely
  - building up ~syntactic structure
  - final linearization
- built for English
  - large coverage lexicon included
  - ports to multiple languages available

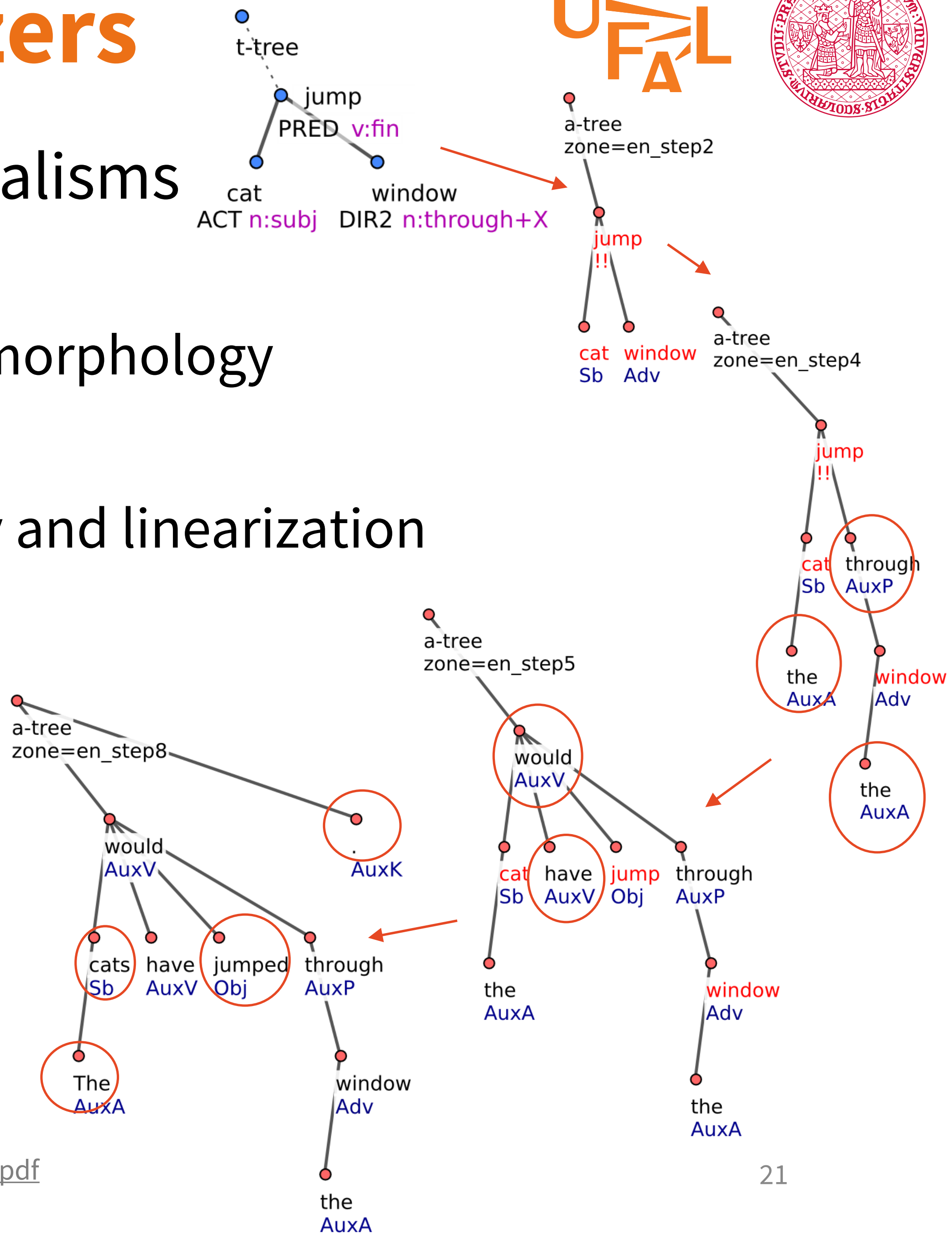
SimpleNLG
generation procedure

```
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
NLGFactory nlgFactory = new NLGFactory(lexicon);
Realiser realiser = new Realiser(lexicon);

SPhraseSpec p = nlgFactory.createClause();

p.setSubject("Mary");
p.setVerb("chase");
p.setObject("the monkey");

p.setFeature(Feature.TENSE, Tense.PAST);

String output = realiser.realiseSentence(p);
System.out.println(output);

>>> Mary chased the monkey.
```

(Gatt & Reiter, 2009)
https://www.aclweb.org/anthology/W09-0613

Grammar/Procedural Realizers

- procedural, but based on grammar formalisms
- **RealPro** (Meaning-Text-Theory)
  - deep syntax/semantics → surface syntax → morphology
- **Treex** (Functional Generative Description)
  - deep syntax → surface syntax → morphology and linearization
  - simple Perl program
    - copy deep syntax
    - fix morphology agreement
    - add prepositions, conjunctions & articles
    - add auxiliary verbs
    - inflect words
    - add punctuation & capitalization



(Lavoie & Rambow, 1997)
http://dl.acm.org/citation.cfm?id=974596

(Popel & Žabokrtský 2010; Dušek et al., 2015)
https://ufal.mff.cuni.cz/~popel/papers/2010_icetal.pdf
https://www.aclweb.org/anthology/W15-3009

Trainable Realizers

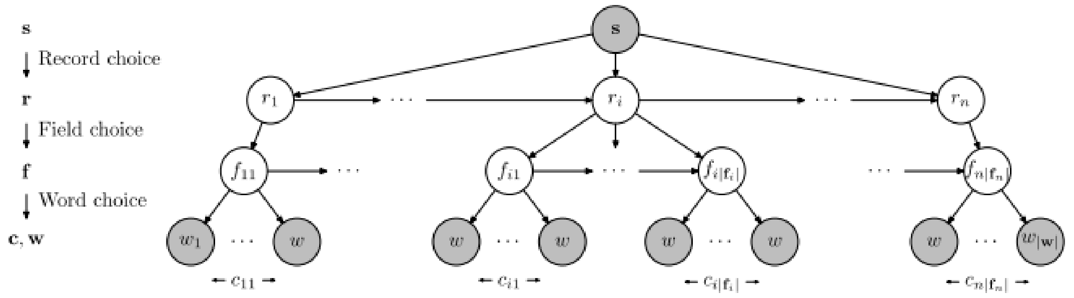
- **Overgenerate & Rerank**
  - same approach as for sentence planning
  - assuming a flexible handcrafted realizer (e.g., OpenCCG)
  - underspecified input → more outputs possible
  - generate more & use statistical reranker, based on:
    - n-gram language models NITROGEN (Langkilde & Knight, 1998) https://www.aclweb.org/anthology/P98-1116 HALOGEN (Langkilde-Geary, 2002) https://www.aclweb.org/anthology/W02-2103
    - Tree language models FERGUS (Bangalore & Rambow, 2000) https://aclweb.org/anthology/C00-1007
    - expected text-to-speech output quality (Nakatsu & White, 2006) https://www.aclweb.org/anthology/P06-1140
    - personality traits & alignment/entrainment CRAG (Isard et al., 2006) https://www.aclweb.org/anthology/W06-1405
  - more variance, but at computational cost

this means the grammar may be smaller

- **Grammar/Procedural-based** StuMaBa (Bohnet et al., 2010) https://www.aclweb.org/anthology/C10-1012
  - same as RealPro or TectoMT, but predict each step using a classifier

Non-Neural End-to-End NLG

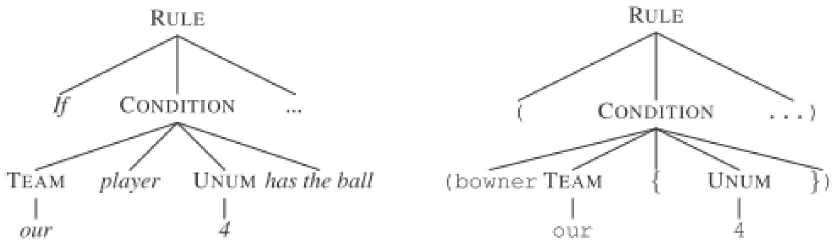
- NLG **as language models**
  - hierarchy of language models (HMM/MEMM/CRF style)
  - DA → slot → word level



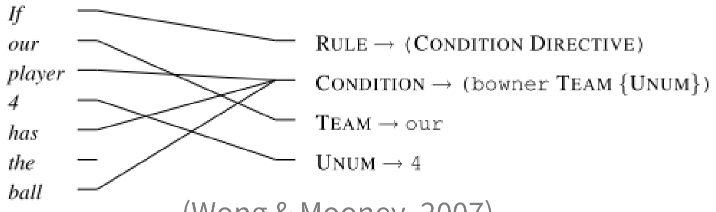
(Oh & Rudnicky, 2002) https://doi.org/10.1016/S0885-2308(02)00012-8
(Angeli et al., 2010) https://www.aclweb.org/anthology/D10-1049
(Liang et al., 2009) https://www.aclweb.org/anthology/P09-1011
(Mairesse et al., 2010) https://www.aclweb.org/anthology/P10-1157
(Mairesse & Young, 2014) https://www.aclweb.org/anthology/J14-4003

- NLG **as parsing**
  a) "language models" by probabilistic CFGs
     - approximate search for best CFG derivation
  b) synchronous PCFGs – MRs & text
     - "translation" with hierarchical phrase-based system
     - parsing MR & generating text

rule	prob./parameter
1. $S \rightarrow R(start)$	$[Pr = 1]$
2. $R(r_i.t) \rightarrow FS(r_j, start)\ R(r_j.t)$	$[P(r_j.t \mid r_i.t) \cdot \lambda]$
3. $R(r_i.t) \rightarrow FS(r_j, start)$	$[P(r_j.t \mid r_i.t) \cdot \lambda]$
4. $FS(r, r.f_i) \rightarrow F(r, r.f_j)\ FS(r, r.f_j)$	$[P(f_j \mid f_i)]$
5. $FS(r, r.f_i) \rightarrow F(r, r.f_j)$	$[P(f_j \mid f_i)]$
6. $F(r, r.f) \rightarrow W(r, r.f)\ F(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
7. $F(r, r.f) \rightarrow W(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
8. $W(r, r.f) \rightarrow \alpha$	$[P(\alpha \mid r, r.f, f.t, f.v)]$
9. $W(r, r.f) \rightarrow g(f.v)$	$[P(g(f.v).mode \mid r, r.f, f.t = int)]$



(a) English (b) CLANG



(Wong & Mooney, 2007)
https://www.aclweb.org/anthology/N07-1022

(Konstas & Lapata, 2012)
https://www.aclweb.org/anthology/P12-1039

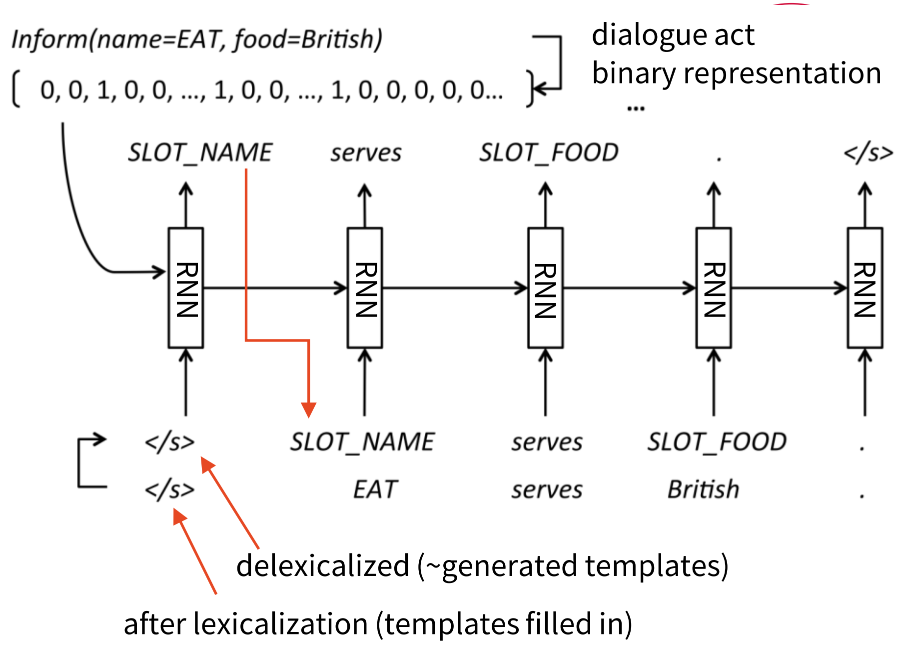
Neural End-to-End NLG: RNNLG

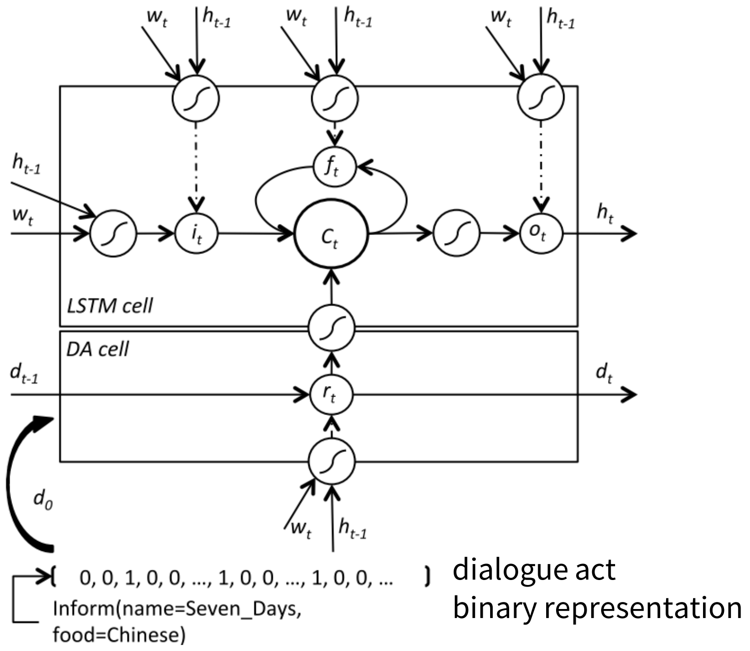
- Unlike previous, doesn’t need alignments
  - no need to know which word/phrase corresponds to which slot

```
name [Loch Fyne], eatType[restaurant], food[Japanese], price[cheap], familyFriendly[yes]
```

Loch Fyne is a kid-friendly restaurant serving cheap Japanese food.

- Using RNNs, generating word-by-word
  - neural language models conditioned on DA
  - generating delexicalized texts
- input DA represented as binary vector
- Enhanced LSTM cells (SC-LSTM)
  - special part of the cell (gate) to control slot mentions



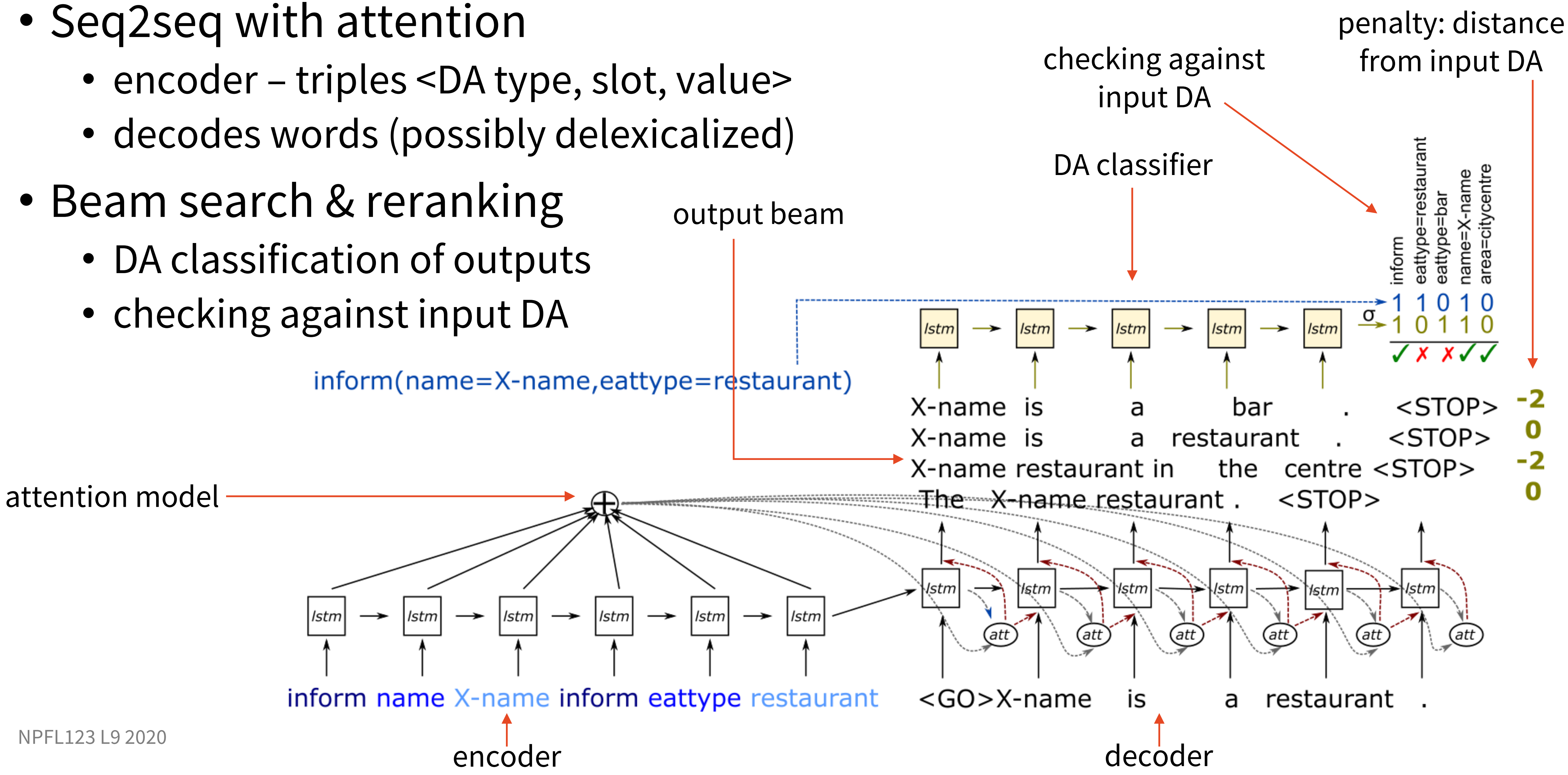


(Wen et al, 2015; 2016) http://aclweb.org/anthology/D15-1199 http://arxiv.org/abs/1603.01232

Seq2seq NLG (TGen)

(Dušek & Jurčíček, 2016)
https://aclweb.org/anthology/P16-2008

- Seq2seq with attention
  - encoder – triples <DA type, slot, value>
  - decodes words (possibly delexicalized)
- Beam search & reranking
  - DA classification of outputs
  - checking against input DA



Problems with neural NLG

(Dušek et al., 2019)
http://arxiv.org/abs/1901.07931

- Checking the semantics
  - neural models tend to forget / make up irrelevant stuff
  - reranking currently best, but not perfect
- Delexicalization needed (at least some slots)
  - otherwise the data would be too sparse
  - alternative: copy mechanisms
- Diversity & complexity of outputs
  - still can't match humans
  - needs specific tricks to improve this
- Still more hassle than writing up templates 😏

open sets, verbatim on the output
(e.g., restaurant/area names)

Summary

- **Deep Reinforcement Learning**
  - same as plain RL – agent + states, actions, rewards – just Q or π is a NN
  - function approximation for Q – mean squared value error
  - **Deep Q Networks** – Q learning where Q is a NN + tricks
    - experience replay, target function freezing
  - **Policy networks** – policy gradients where π is a NN

- **Natural Language Generation**
  - steps: **content planning**, **sentence planning**, **surface realization**
    - not all systems implement everything (content planning is DM's job in DS)
    - pipeline vs. end-to-end
  - approaches: templates, grammars, statistical
  - templates work great
  - state-of-the-art = seq2seq with reranking

Thanks

Contact us:

odusek@ufal.mff.cuni.cz
hudecek@ufal.mff.cuni.cz
Slack

Get these slides here:

http://ufal.cz/npfl123

References/Inspiration/Further:

- Matiisen (2015): Demystifying Deep Reinforcement Learning: https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
- Karpathy (2016): Deep Reinforcement Learning – Pong From Pixels: http://karpathy.github.io/2016/05/31/rl/
- David Silver's course on RL (UCL): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Sutton & Barto (2018): Reinforcement Learning: An Introduction (2nd ed.): http://incompleteideas.net/book/the-book.html
- Milan Straka's course on RL (Charles University): http://ufal.mff.cuni.cz/courses/npfl122/
- Deep RL for NLP tutorial
- Mnih et al. (2013): Playing Atari with Deep Reinforcement Learning: https://arxiv.org/abs/1312.5602
- Mnih et al. (2015): Human-level control through deep reinforcement learning: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- Gatt & Krahmer (2017): Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation http://arxiv.org/abs/1703.09902
- My PhD thesis (2017), especially Chapter 2: http://ufal.mff.cuni.cz/~odusek/2017/docs/thesis.print.pdf